Architecture

Eng1 Group 29

Adam Hewlett
Ani Thomas
Dan Kirkpatrick
Dominik Hagowski
Matthew Crompton
Niko Chen

Architecture

Representation of the Architecture

We opted to use both a Behavioural Sequence Diagram[1](Chart1) and a Structural Class Diagram[2](Chart 2) in our representation of the architecture. We chose each diagram as they would simply and visually illustrate both the flow of the gameplay and to more rigidly aid with the implementation in the case of the Class diagram.

Language and tool used in order to model the architecture

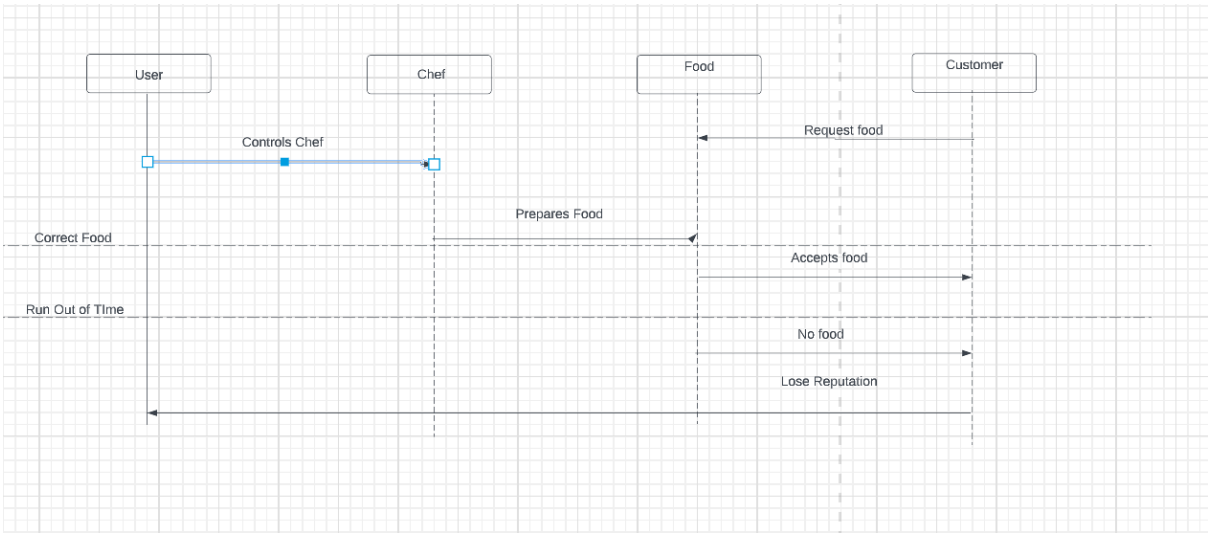
We chose to use UML in order to best represent our architecture for several key reasons:

- It suited our choice of Object-Oriented-Programming
- It offers easy to follow visuals when the team moves on to the implementation
- In a similar vein as the last point, it allows for non-team members to easily and visually understand our thought processes

To facilitate the creation of these diagrams, we opted to use 2 key tools, SmartDraw and LucidChart.

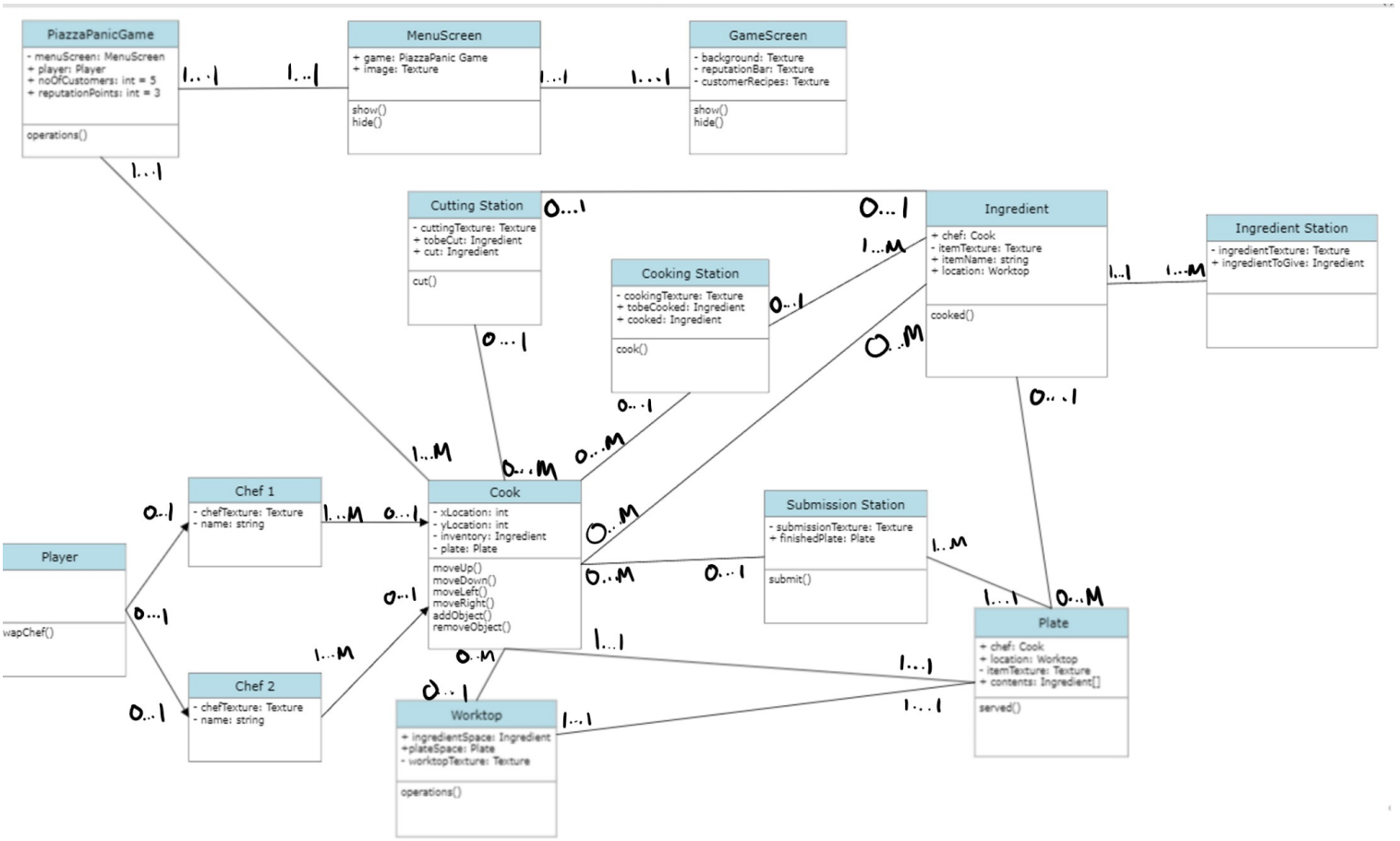
- SmartDraw offered the easiest interface with which to create Structural Class Diagrams but had the drawback of being a usually paid service. We however found that in this case, the productivity increase offered by SmartDraw was well worth the extra effort.
- LucidChart was used primarily to create our Behavioural Sequence Diagram. It had the advantage over SmartDraw in this use-case as its interface lends itself to creating a more abstract diagram. LucidChart also offers far more options to a free user in comparison to SmartDraw which made it an obvious alternative for this case.

Behavioural Sequence Diagram

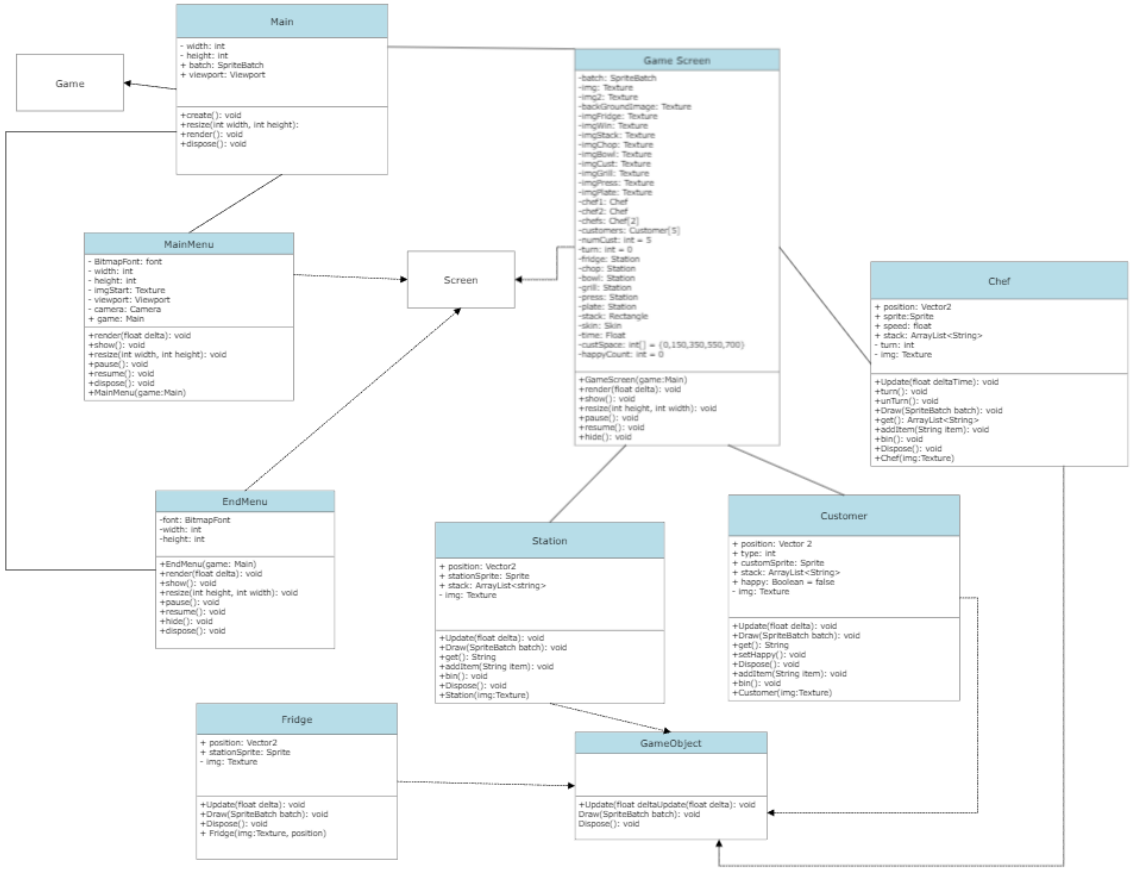


Although our initial behavioural diagram is very simple, it allowed the team to understand the most basic interactions that the player must be able to undertake. The creation of this behavioural diagram was vital in the creation of our structural diagram which offers far more assistance on how the project is to be implemented.

Initial Structural Class Diagram



Final Structural Class Diagram



Note - only the Main to Game relationship is inheritance. The rest of the arrows are dotted but this may not be entirely clear.

Systematic justification for the design of the architecture

Our choice to default to Object-Oriented-Programming for this project was primarily due to the team all already being incredibly familiar with Object-Oriented-Programming. We however did consider another software architecture pattern frequently used in game design, Entity Component Design[3]. We made the following comparisons between the two:

- An ECS(Entity Component System) is less well known and thus has less documentation present in order for us to learn to effectively utilise it
- ECS provides a major advantage in large projects such as games typically however in our case, the simplicity of the game (there are not a huge amount of classes and inheritances) minimises the effectiveness of ECS over OOP/
- There is no extra-work needed to be done by the team in order to learn the pattern and therefore saving time.

Behavioural Sequence Diagram Use

This provided a high-level overview of the very basic functionality of the game. This was used in order to create a lower-level Structural Class Diagram which in turn sped up our implementation. This diagram was also used throughout implementation in order to test the overall functionality and flow of the game

Structural Class Diagram Use

The structural class diagram provided a more detailed plan as to how the project should have been implemented. Although we ended up deferring from the original diagram, it provided a solid basis from which we were able to improve our architecture.

The evolution of our Structural Diagram from the Initial to the Concrete

Our Structural Class Diagram underwent a huge amount of change between its current iteration and the original to the point where the two seem unrelated. The major differences came very early when we realised we would need to rework our structure after spending further time with libGDX in order to implement the project. The inclusions of the Game and Screen Classes that are baked into libGDX are included to show the true architecture. One of the primary changes is the fact that all the worktops, cooking stations and chopping stations were merged under a single Station Class and are handled by the GameScreen class as opposed to how they were very spread apart in the initial. The quality and accuracy of the diagram also was improved, with far more operations and attributes present to more accurately represent the full implementation. This structural diagram will work in conjunction with concise and clear commenting to better allow non-team members to understand exactly how the code is structured.

Our final concrete structural class diagram has severely cut down on the number of classes and relations to create a simpler to understand system that has the added bonus of being easy to read by non-team members.

Relation of the Architecture to the Requirements

Each of the classes in the concrete architecture served to fulfil the requirements we had gained from the client.

- GameScreen - Provided most of the visuals for the game as well as setting up the environment [**FR_UI**], [**FR_AREA_BOUNDARIES**]
- Chef - The character that the player is able to control to interact with the environment [**FR_Cook_Move**], [**FR_Cook_Interact**]
- Station - Provided the the functionality for the cutting, cooking and ingredient stations [**FR_Cook_Interact**], [**FR_PREP_STATION**], [**FR_SERVE**]
- Customer - Hold the recipes that need to be catered to [**FR_UI**], [**FR_RANDOM**], [**FR_CUSTOMER**]
- Fridge - Allows the Chef to pick up the necessary ingredients [**FR_PREP_STATION**]
- MainMenu - creates the Main Menu for the game and allows the player to press a key to continue [**FR_TITLE_SCEEN**]

Bibliography

1. https://sparxsystems.com/enterprise_architect_user_guide/16.1/modeling_languages/behavioraldiagrams.html#:~:text=UML%20Behavioral%20Diagrams%20depict%20the,convey%20the%20passage%20of%20time.
2. https://www.lucidchart.com/pages/uml-class-diagram

3. https://www.simplilearn.com/entity-component-system-introductory-guide-article#:~:text=OOP%20encourages%20data%20encapsulation%20while,separates%20the%20data%20from%20behavior
4. https://creately.com/guides/class-diagram-relationships/#:~:text=Multiplicity,-Multiplicity&text=is%20the%20active%20logical%20association,contain%20zero%20to%20many%20passengers.